CSC 228-01 Data Structures and Algorithms, Fall 2018
Instructor: Dr. Natarajan Meghanathan

Exam 1 (Take Home)

Submission: Submit everything together as one PDF file in Canvas. Due: Oct. 4th, by 11.59 PM

Q1 - 20 pts) Consider the implementation of the List ADT using Singly Linked List given to you for this question. Add a member function (to the List class) called *recursivePrintForwardReverseOrders* that prints the contents of the list in a recursive fashion in both the forward order and reverse order.

For example, if the contents of the List are: 10 --> 4 --> 8 --> 12 --> 9, the recursive member function should print the List as follows:
10 4 8 12 9
9 12 8 4 10

Note that both the forward and reverse orders should be printed through an invocation of the *recursivePrintForwardReverseOrders* member function on the List object called from the main function. You are free to choose the parameter(s) that need to be passed to the *recursivePrintForwardReverseOrders* function. But, you are not supposed to pass more than three parameter(s). A suggestion for the parameter to pass is given in the main function of the code posted for Question 1.

To test your code (and take screenshot), create a List of at least 10 elements (with a maximum value of 100) and then call the *recursivePrintForwardReverseOrders* function on this List object by passing a pointer to the first node in the Linked List as an argument, as shown in the main function of the Singly Linked List code for Question 1.

You need to submit the following as part of your answer for this question:
(i) the complete code for the Node class, List class (including the recursivePrintForwardReverseOrders( ) function) and the main function.
(ii) Snapshot of the execution of the code with at least 10 elements and maximum value of 100.

Q2 - 15 pts)
Consider the implementation of the Singly Linked List class (named: List) given to you for this question. Your task is to add a member function called pairwiseSwap( ) to the Singly Linked List class such that it can be called from the main function (as given in the code) to swap the elements of an integerList (an object of the class List) pairwise and print the updated list. For example, if the List before the pairwiseSwap is 4 -> 5 -> 2 -> 3 -> 1 -> 6, after the pairwiseSwap, the contents of the list should be: 5 -> 4 -> 3 -> 2 -> 6 -> 1. If the List has an odd number of elements, like: 4 -> 5 -> 2 -> 3 -> 1, then after the pairwiseSwap, the contents of the list should be: 5 -> 4 -> 3 -> 2 -> 1.

Test your code with 10 elements and 11 elements (with a maximum value of 25 in each case) and take screenshots of the List before and after pairwiseSwap, as printed in the main function.

You need to submit the following as part of your answer for this question:
(i) the complete code for the Node class, List class (including the pairwiseSwap( ) function) and the main function.
(ii) Snapshots of the execution of the code with 10 elements and 11 elements (with a maximum value of 25 in each case).

Q3 - 15 pts)
The deleteElement(int deleteData) member function in the code for the Singly Linked List-based implementation of a List ADT deletes the first occurrence of deleteData in the List. Modify this member function in such a way that all occurrences of deleteData in the List are deleted with a single call to the deleteElement function from main. After you modify the *deleteElement* function, run the main function (as given in the startup code for this question) by creating a List of at least 15 elements (with a maximum value of 10 for any element so that certain elements repeat). Now ask for a value (deleteData) to delete from the user and call the deleteElement(deleteData) function to delete all occurrences of deleteData in the List. Capture the output of your program displaying the contents of the List before and after the call to the deleteElement function.

You need to submit the following as part of your answer for this question:
(i) the complete code for the Node class, List class (including the modified version of the deleteElement(deleteData) function) and the main function.
(ii) Screenshot of the execution of the code as mentioned above (list of 15 elements, with a maximum value of 10 so that some elements repeat and you try to delete one of such repeating elements).

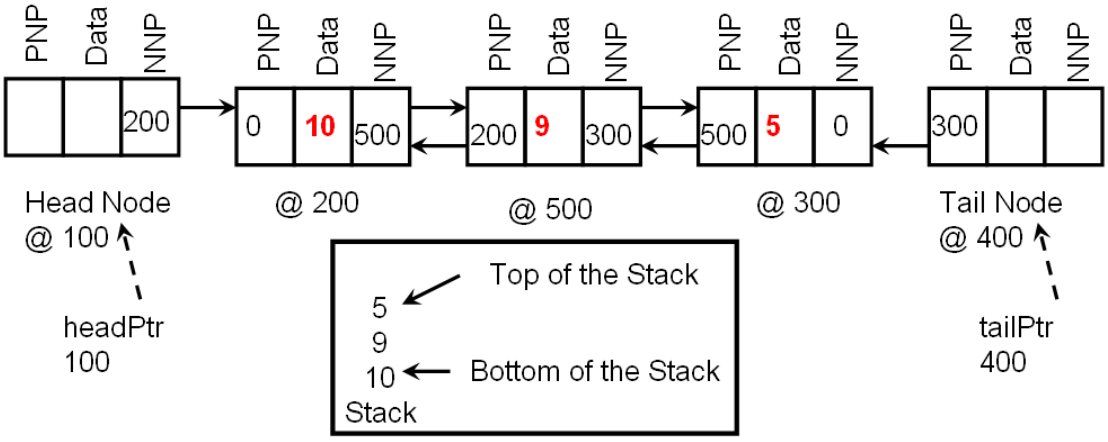
Q4 - 25 pts)
Implement Stack ADT as a Singly Linked List without using the insertAtIndex, deleteElement, readIndex functions of the Singly Listed List. That is, the push, pop and peek operations should not call insertAtIndex(0, data), deleteElement(0) and readIndex(0) functions. The push, pop and peek operations should be directly implemented to insert an element in the beginning of the linked list, to delete an element from the beginning of the linked list and to read the element value from the beginning of the linked list. To help you out, the implementation of the push function is given in the startup code provided. Your task is to implement the peek and pop functions like this without calling any other function. You can notice that the insertAtIndex, deleteElement and readIndex functions have been removed from the Stack class and you should not use them.

After implementing the pop and peek functions, you will be comparing the actual run-time of the push and pop operations of a Stack implemented as a Singly Linked List (with insertions and deletions in the beginning of the Linked List) with that of a Stack implemented as a Doubly Linked List (with insertions and deletions at the tail/end of the Linked List). The main function provided to you has the timers setup for this purpose. Your task is to just run the main functions and measure the average time taken (in microseconds) for the push and pop operations with the Stack as a Singly Linked List and with the Stack as a Doubly Linked List for the following values of the parameters: (i) # elements to be pushed = 1000, 10000, 100000, 1000000; (ii) maximum value for any element = 50000 and (iii) # trials = 50.

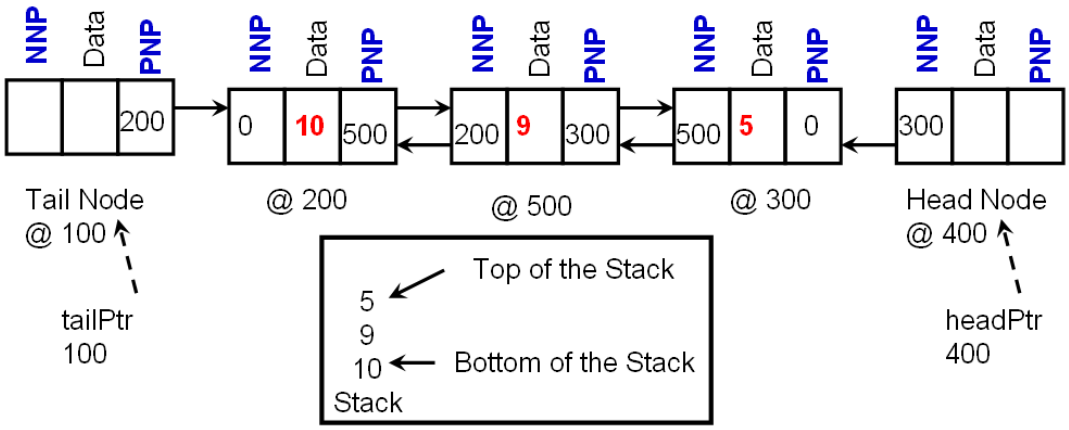
You need to include the following as part of your answer to this question:
(i) the code for the Singly Linked List-based implementation of the Stack class
(ii) a table presenting the actual run-times for the parameters mentioned above
(iii) snapshots of the actual run-times for the above cases.
(iv) interpretation of the difference/similarity in the actual run-times for the push and pop operations between the Singly Linked List and Doubly Linked List implementation of the Stack ADT.

Q5 - 25 pts)
Consider the Doubly Linked List-based implementation of the Stack ADT, the code for which is given to you and also discussed in class. According to this code, the latest data to be pushed to the Stack is inserted at the tail/end of the list and the earliest pushed data is in the bottom of the stack. This implies that (as per the code given to you) the node next to the head node has its data at the bottom of the stack and the node previous to the tail node has its data at the top of the stack. The IterativePrint( ) function given in the code prints the contents of the Stack from the head node to the tail node.

The following figure illustrates a Stack with elements pushed in this order: 10, 9, 5 and a call to the IterativePrint( ) function will also print the elements in this order. Note that the abbreviations PNP and NNP in the figure refer to PrevNodePtr and NextNodePtr respectively.



In this question, you are required to reverse the links of the Doubly Linked List-based implementation of the Stack such that the tailPtr points to the head node (and hence the head node has to be now appropriately referred to as the tail node) and the headPtr points to the tail node (and hence the tail node has to be now appropriately referred to as the head node), as well as the values for the prevNodePtr and nextNodePtr pointers for every node (including the head node and tail node) swapped between themselves. With all these link reversals, the node next to the head node has its data at the top of the stack and the node previous to the tail node has its data at the bottom of the stack. A call to the IterativePrint( ) function (that is designed to print the Stack from the head node to the tail node) will now print the contents of the Stack from the top to the bottom (as the next node of the head node has its data now at the top of the stack) as: 5, 9, 10. The Stack of the example shown above will look like the figure below after the link reversals. Note that the values for the PNP and NNP are swapped as highlighted (e.g., for the node with data 10, the PNP and NNP were respectively 0 and 500 before the link reversals, and are 500 and 0 respectively, after the link reversals).



You need to include the following as part of your answer to this question:

(i) the entire code for the Doubly linked list-based implementation of the Stack class, including the reverseStack( ) function.

(ii) snapshot of the execution of the code for list size of 10 and maximum value for each element being 100.